МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Искусственные нейронные сети»
Тема: «Распознавание объектов на фотографиях»

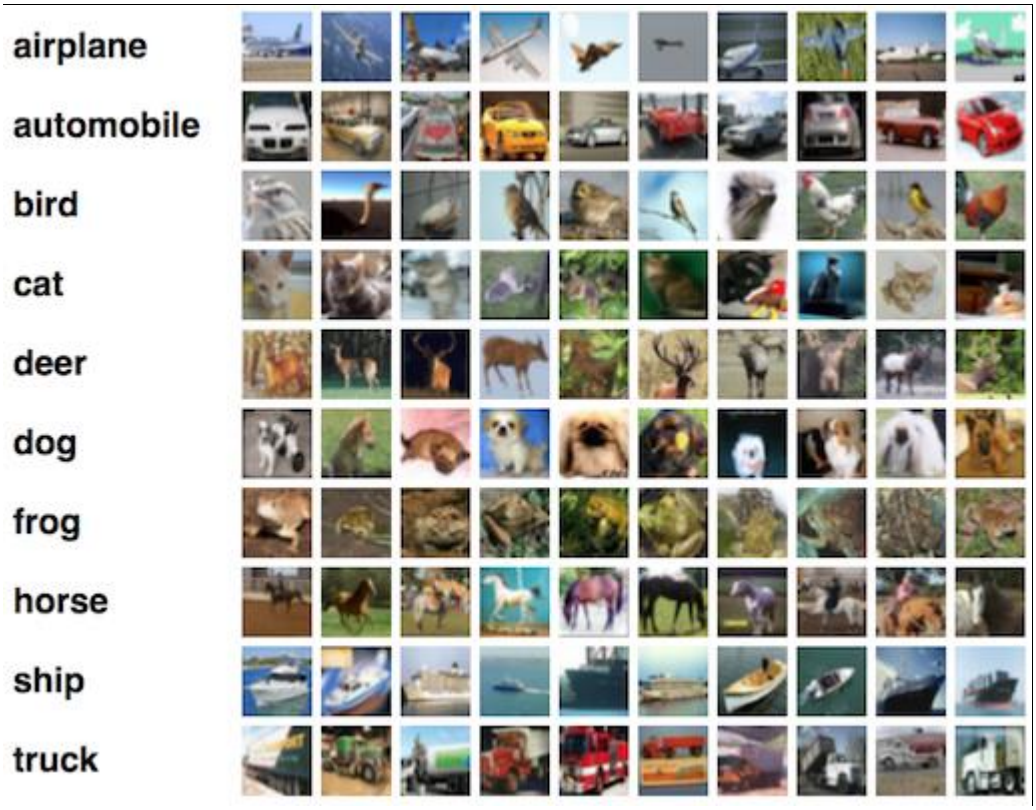
Студент гр. 7381 _____ Лукашев Р.С.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

Цель работы.

Распознавание объектов на фотографиях (Object Recognition in Photographs) CIFAR-10 (классификация небольших изображений по десяти классам: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль и грузовик).



Задачи.

- Ознакомиться со сверточными нейронными сетями;
- Изучить построение модели в Keras в функциональном виде;
- Изучить работу слоя разреживания (Dropout);

Требования.

1. Построить и обучить сверточную нейронную сеть;
2. Исследовать работу сеть без слоя Dropout;
3. Исследовать работу сети при разных размерах ядра свертки.

Ход работы.

Была написана функция, создающая модель с заданным размером ядра свертки, и опционально с dropout слоями. Вместо 200 эпох было выбрано 15, поскольку такого количества эпох достаточно, чтобы делать какие-то выводы и сравнивать модели. Посмотрим на разницу между моделью, в которой есть слои разреживания, и в которой таковые отсутствуют (см. рис 1, 2).

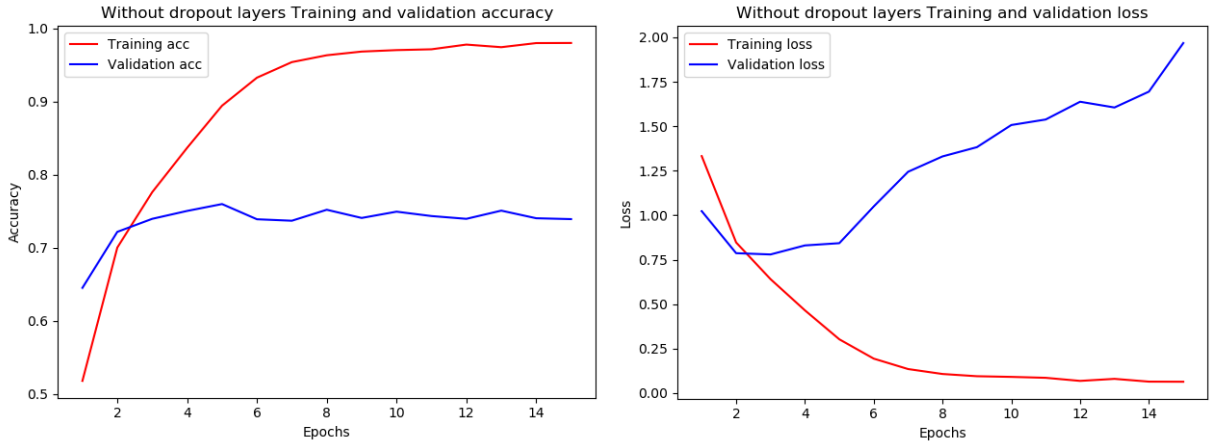


Рис. 1 – Точность и потери при обучении модели без слоев разреживания.

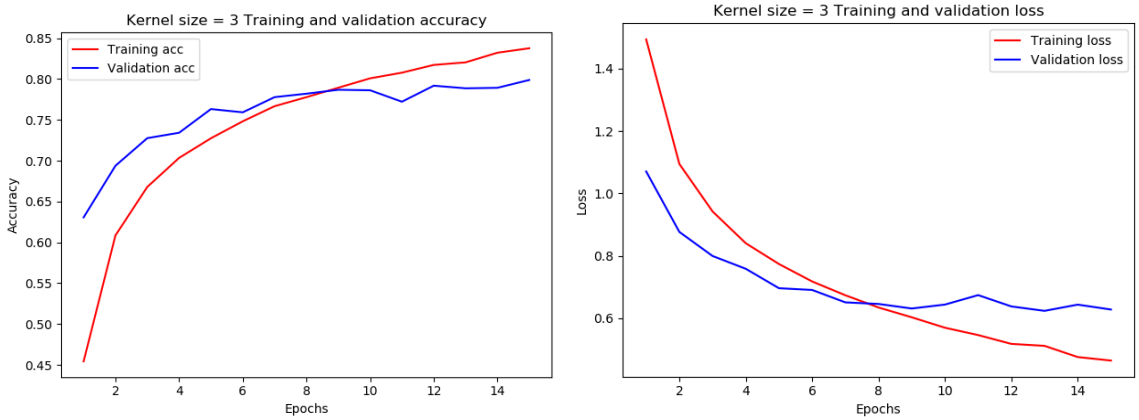


Рис. 2 – Точность и потери при обучении модели со слоями разреживания.

Как можно заметить, у модели без слоев разреживания быстро происходит переобучение.

Далее посмотрим, что случится, если мы изменим размер ядра свертки (см. рис 3).

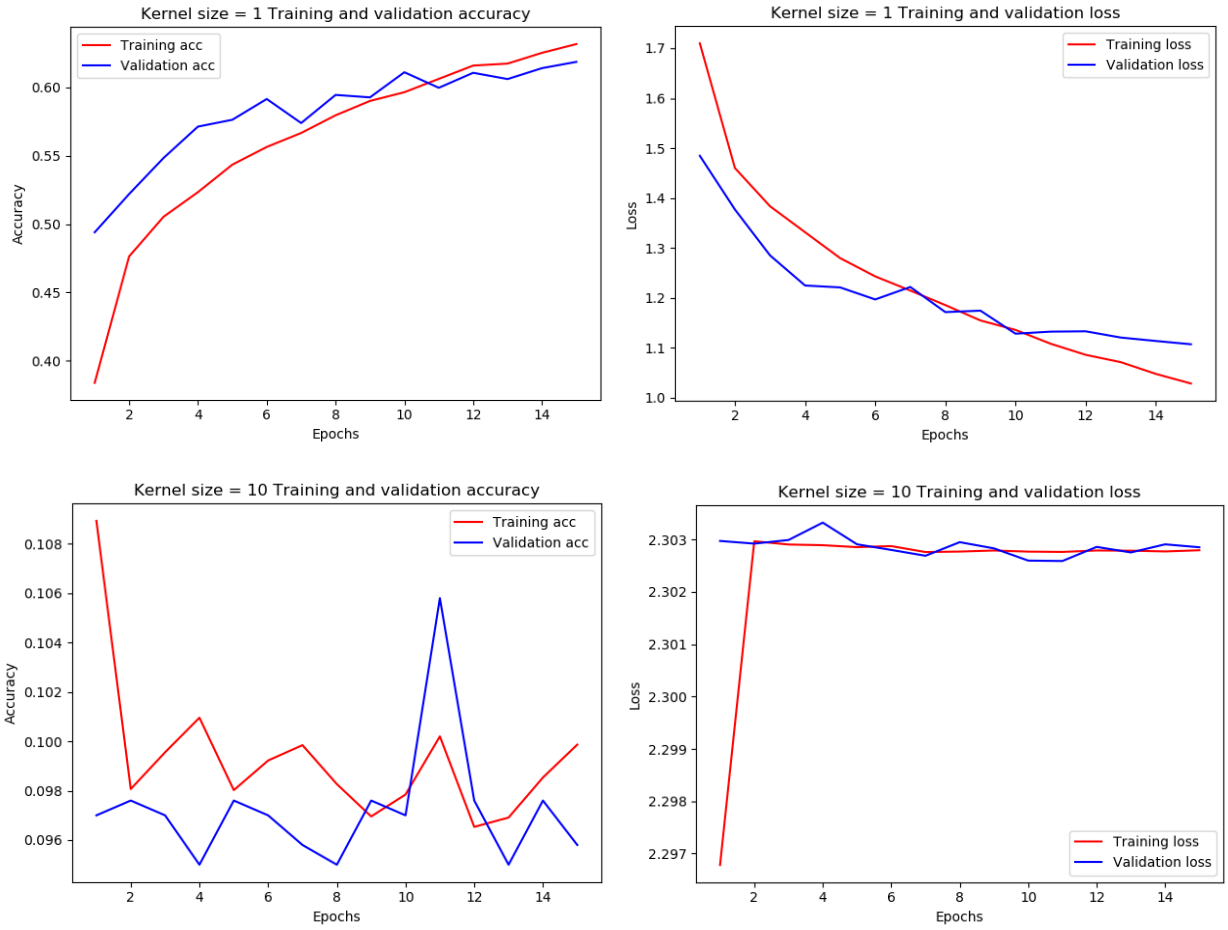


Рис. 3 – Точность и потери при обучении моделей с разным размером ядра.

Размерность ядра контролирует глубину тензора признаков. Результатом действия каждого сверточного уровня является карта признаков. Маленькие размерности (3x3) позволяют улавливать более мелкие детали, что в данном случае позволяет точнее классифицировать картинку. Ядро размерностью 1x1 улавливает слишком мелкие признаки, которые вполне могут принадлежать всем или нескольким классам одновременно, из-за чего уменьшилась точность классификации. Большие размерности не способны улавливать мелкие детали, что в данном случае привело к невозможности обучить модель.

Выводы.

В ходе выполнения данной лабораторной работы было произведено ознакомление со сверточными нейронными сетями: изучено построение модели в Keras

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
from keras.datasets import cifar10
from keras.models import Model
from keras.layers import Input, Convolution2D, MaxPooling2D, Dense, Dropout, Flatten
from keras.utils import np_utils
import matplotlib.pyplot as plt
import numpy as np


batch_size = 32 # in each iteration, we consider 32 training examples at once
num_epochs = 15 # we iterate 15 times over the entire training set
kernel_size = [1, 3, 10] # we will use 3x3 kernels throughout
pool_size = 2 # we will use 2x2 pooling throughout
conv_depth_1 = 32 # we will initially have 32 kernels per conv. layer...
conv_depth_2 = 64 # ...switching to 64 after the first pooling layer
drop_prob_1 = 0.25 # dropout after pooling with probability 0.25
drop_prob_2 = 0.5 # dropout in the dense layer with probability 0.5
hidden_size = 512 # the dense layer will have 512 neurons


(X_train, y_train), (X_test, y_test) = cifar10.load_data() # fetch CIFAR-10 data


num_train, depth, height, width = X_train.shape # there are 50000 training examples
in CIFAR-10
num_test = X_test.shape[0] # there are 10000 test examples in CIFAR-10
num_classes = np.unique(y_train).shape[0] # there are 10 image classes


X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= np.max(X_train) # Normalise data to [0, 1] range
X_test /= np.max(X_train) # Normalise data to [0, 1] range


Y_train = np_utils.to_categorical(y_train, num_classes) # One-hot encode the labels
Y_test = np_utils.to_categorical(y_test, num_classes) # One-hot encode the labels
```

```
def fit_model(kernel_size=3, dropout=True):

    inp = Input(shape=(depth, height, width))  # N.B. depth goes first in Keras


    # Conv [32] -> Conv [32] -> Pool (with dropout on the pooling layer)

    conv_1 = Convolution2D(conv_depth_1, kernel_size, kernel_size,
border_mode='same', activation='relu')(inp)

    conv_2 = Convolution2D(conv_depth_1, kernel_size, kernel_size,
border_mode='same', activation='relu')(conv_1)

    pool_1 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_2)

    if dropout:

        drop_1 = Dropout(drop_prob_1)(pool_1)

    else:

        drop_1 = conv_2

    # Conv [64] -> Conv [64] -> Pool (with dropout on the pooling layer)

    conv_3 = Convolution2D(conv_depth_2, kernel_size, kernel_size,
border_mode='same', activation='relu')(drop_1)

    conv_4 = Convolution2D(conv_depth_2, kernel_size, kernel_size,
border_mode='same', activation='relu')(conv_3)

    pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_4)

    if dropout:

        drop_2 = Dropout(drop_prob_1)(pool_2)

    else:

        drop_2 = conv_4


    # Now flatten to 1D, apply Dense -> ReLU (with dropout) -> softmax

    flat = Flatten()(drop_2)

    hidden = Dense(hidden_size, activation='relu')(flat)

    if dropout:

        drop_3 = Dropout(drop_prob_2)(hidden)

    else:

        drop_3 = hidden


    out = Dense(num_classes, activation='softmax')(drop_3)


    model = Model(input=inp, output=out)  # To define a model, just specify its input
and output layers
```

```
    model.compile(loss='categorical_crossentropy',  # using the cross-entropy loss
function
                  optimizer='adam',  # using the Adam optimiser
                  metrics=['accuracy'])  # reporting the accuracy


    history = model.fit(X_train, Y_train,  # Train the model using the training
set...
              batch_size=batch_size, nb_epoch=num_epochs,
              verbose=1, validation_split=0.1)  # ...holding out 10% of the data for
validation


    return history



def graph(H, title):
    loss = H.history['loss']
    val_loss = H.history['val_loss']
    acc = H.history['accuracy']
    val_acc = H.history['val_accuracy']
    epochs = range(1, len(loss) + 1)


    plt.title(title)
    # Построение графика ошибки
    plt.plot(epochs, loss, 'r', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title(title + ' Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()


    # Построение графика точности
    plt.clf()
    plt.plot(epochs, acc, 'r', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title(title + ' Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
```

```
    plt.legend()
    plt.show()


graph(fit_model(kernel_size=3, dropout=False), "Without dropout layers")

for k_s in kernel_size:
    graph(fit_model(kernel_size=k_s, dropout=True), "Kernel size = " + str(k_s))
```